SIAM J. SCI. COMPUT.
Vol. 30, No. 2, pp. 764–784

A FRAMEWORK FOR DISCRETE INTEGRAL TRANSFORMATIONS I—THE PSEUDOPOLAR FOURIER TRANSFORM[*]

A. AVERBUCH[†], R. R. COIFMAN[‡], D. L. DONOHO[§], M. ISRAELI[¶], AND Y. SHKOLNISKY[‡]

In memory of Moshe Israeli 1940–2007

Abstract. The Fourier transform of a continuous function, evaluated at frequencies expressed in polar coordinates, is an important conceptual tool for understanding physical continuum phenomena. An analogous tool, suitable for computations on discrete grids, could be very useful; however, no exact analogue exists in the discrete case. In this paper we present the notion of pseudopolar grid (pp grid) and the pseudopolar Fourier transform (ppFT), which evaluates the discrete Fourier transform at points of the pp grid. The pp grid is a type of concentric-squares grid in which the radial density of squares is twice as high as usual. The pp grid consists of equally spaced samples along rays, where different rays are equally spaced in slope rather than angle. We develop a fast algorithm for the ppFT, with the same complexity order as the Cartesian fast Fourier transform; the algorithm is stable, invertible, requires only one-dimensional operations, and uses no approximate interpolations. We prove that the ppFT is invertible and develop two algorithms for its inversion: iterative and direct, both with complexity $O(n^2 \log n)$, where $n \times n$ is the size of the reconstructed image. The iterative algorithm applies conjugate gradients to the Gram operator of the ppFT. Since the transform is ill-conditioned, we introduce a preconditioner, which significantly accelerates the convergence. The direct inversion algorithm utilizes the special frequency domain structure of the transform in two steps. First, it resamples the pp grid to a Cartesian frequency grid and then recovers the image from the Cartesian frequency grid.

Key words. unequally spaced FFT, pseudopolar Fourier transform, polar Fourier transform, fractional Fourier transform, concentric-squares grid, linogram

AMS subject classification. 65T50

DOI. 10.1137/060650283

1. Introduction.

Given a function $f(x, y)$ of a continuous argument $(x, y) \in \mathbb{R}^2$, its two-dimensional (2D) Fourier transform, denoted $\hat{f}(\omega_x, \omega_y)$, is given by the integral

$$\hat{f}(\omega_x, \omega_y) = \int_{\mathbb{R}^2} f(x, y) e^{-2\pi i(x\omega_x + y\omega_y)}\, dx\, dy, \qquad \omega_x, \omega_y \in \mathbb{R}. \tag{1}$$

For discrete images $I(u, v)$, $-n/2 \le u, v < n/2$, the corresponding concept is the sum

$$\hat{I}(\omega_x, \omega_y) = \sum_{u,v=-n/2}^{n/2-1} I(u, v) e^{-\frac{2\pi i}{m}(u\omega_x + v\omega_y)}, \quad \omega_x, \omega_y \in \mathbb{R}, \tag{2}$$

where we depart from tradition by letting $m \ge n$ be an arbitrary integer. We assume for simplicity that the image I has equal extent in the x and y directions and that n

[*]Received by the editors January 18, 2006; accepted for publication (in revised form) July 2, 2007; published electronically February 14, 2008.
http://www.siam.org/journals/sisc/30-2/65028.html

[†]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (amir@math.tau.ac.il).
[‡]Department of Mathematics, Yale University, New Haven, CT 06520 (coifman@math.yale.edu, yoel.shkolnisky@yale.edu). The fifth author was supported in part by a grant from the Ministry of Science, Israel.
[§]Statistics Department, Stanford University, Stanford, CA 94305 (donoho@stat.stanford.edu).
[¶]Formerly, Faculty of Computer Science, Technion, Haifa 32000, Israel (israeli@cs.technion.ac.il).

is even. For practical applications, we need to evaluate $\hat{I}$ for (ω_x, ω_y) in some discrete set.

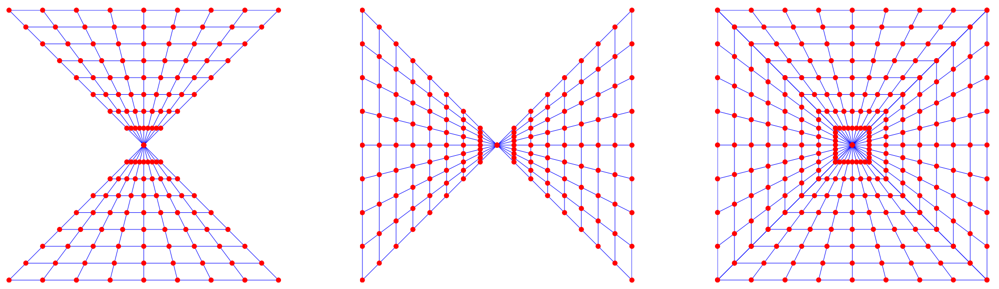
The algorithm presented in this paper precisely evaluates $\hat{I}$ on a special nonuniform point set, the pseudopolar grid (pp grid). It has the same order of complexity as the traditional 2D fast Fourier transform (FFT) for the Cartesian grid and requires only vector operations taking one-dimensional (1D) arrays to 1D arrays, being thereby extremely efficient in today's commercially predominant hierarchical memory machines. At its core, the algorithm reduces to a large number of 1D FFTs and is especially fast in environments where such FFTs are specially optimized. The algorithm does not require an accuracy parameter, and the resulting samples have machine accuracy. This is in contrast to approaches such as [14, 19] that require an accuracy parameter, which controls both the relative accuracy and the processor timing. Because it offers closed-form evaluation on a particular choice of point set, our algorithm is a counterpart of [7, 5, 33], addressing the 2D setting of the pp grid. Those related algorithms work in one dimension, exploiting algebraic properties of the trigonometric polynomial

$$\hat{I}_1(\omega) = \sum_{u=-n/2}^{n/2-1} I_1(u) e^{-2\pi i u \omega / m}$$

to evaluate it in closed form on particular 1D grids: [7] samples $\hat{I}_1$ on points that are equally spaced on the unit circle from 0 to 2π, [5] samples $\hat{I}_1$ on points that are equally spaced on an arbitrary arc of the unit circle, and [33] samples $\hat{I}_1$ on spirals of the form AW^k, where $A, W \in \mathbb{C}$. Like those algorithms, the algorithm we discuss exploits special algebraic properties of the underlying trigonometric polynomial; in those algorithms the polynomial was $\hat{I}_1$, while in this paper it is $\hat{I}$ of (2). Our approach differs from generic nonuniform FFT algorithms, which can work with arbitrary point sets and exploit approximations and expansions to obtain approximate values of $\hat{I}$ on such point sets. For a survey of such nonuniform FFT algorithms, see [35].

The specific point set we use is closely related to the so-called concentric-squares grid from computed tomography [29, 31, 11, 12, 25], as we explain later, but with a higher density of sampling. The extra density gives the resulting transform better properties, for example, with regard to invertibility and geometric fidelity. These properties are crucial for our intended applications.

Indeed numerous tasks in discrete image processing pose problems that seem simple in the continuous setting but which seem awkward using existing tools in the discrete setting. Thus, in the continuous setting several very important operations can be expressed simply and transparently in terms of operations on the Fourier transform in polar coordinates. Examples include image rotation—a shift of the transform in polar coordinates—and Radon transform—the inverse transform of a radial slice. These transparent representations in the continuous setting inspire us to seek correspondingly simple and natural representations in the discrete image processing setting. However, in the discrete domain, polar coordinates and rotations are not intrinsic. We argue below, and have elsewhere shown in several concrete projects, that the pp grid developed here and the corresponding fast algorithms provide an engine to build such transparent representations. For example, our transform obeys the projection-slice theorem connecting Fourier analysis to Radon transform, and it behaves naturally under shearing, which is a natural affine transform of a discrete grid and in some sense the correct substitute for rotation in the discrete setting.



(a) Pseudopolar sector Ω^1_{pp} (b) Pseudopolar sector Ω^2_{pp} (c) Pseudopolar grid $\Omega_{pp} = \Omega^1_{pp} \cup \Omega^2_{pp}$

FIG. 1. *Pseudopolar grid.*

Contents. In section 2 we present the 2D pp grid and the 2D pseudopolar Fourier transform (ppFT). Relationships to previous work and our contribution are considered in section 3. In section 4 we present a fast $O(n^2 \log n)$ algorithm for computing the ppFT. In section 5 we prove the ppFT is invertible, and in sections 6 and 7 we present two efficient algorithms that compute the inverse transform to within an arbitrary prescribed accuracy ε. Section 6 presents an iterative inversion algorithm based on conjugate gradients, while section 7 presents a direct algorithm that is based on fast resampling of the frequency domain.

2. Pseudopolar grid.

The 2D pp grid, denoted Ω_{pp}, is given by

$$\Omega_{pp} \triangleq \Omega^1_{pp} \cup \Omega^2_{pp}, \tag{3}$$

where

$$\Omega^1_{pp} \triangleq \left\{ \left(-\frac{2l}{n}k, k \right) \;\middle|\; -n/2 \le l \le n/2,\ -n \le k \le n \right\}, \tag{4}$$

$$\Omega^2_{pp} \triangleq \left\{ \left(k, -\frac{2l}{n}k \right) \;\middle|\; -n/2 \le l \le n/2,\ -n \le k \le n \right\}. \tag{5}$$

See Figures 1(a), 1(b), and 1(c) for an illustration of Ω^1_{pp}, Ω^2_{pp}, and Ω_{pp}, respectively. As can be seen from the figures, k serves as a "pseudo radius" and l serves as a "pseudo angle." We denote a specific point in Ω^1_{pp} and Ω^2_{pp} by $\Omega^1_{pp}(k,l)$ and $\Omega^2_{pp}(k,l)$, respectively.

The resolution of the pp grid, given by (3)–(5), is $n+1$ in the angular direction and $m = 2n+1$ in the radial direction. The presented construction uses these angular and radial resolutions to support the future derivation of the discrete Radon transform [3]. However, the construction can be repeated with arbitrary resolutions in the angular and radial directions.

In polar coordinates, the pp grid is given by

$$\Omega^1_{pp}(k,l) = (r^1_k, \theta^1_l), \quad \Omega^2_{pp}(k,l) = (r^2_k, \theta^2_l), \tag{6}$$

where

$$r_k^1 = k\sqrt{4\left(\frac{l}{n}\right)^2 + 1}, \quad r_k^2 = k\sqrt{4\left(\frac{l}{n}\right)^2 + 1}, \tag{7}$$

$$\theta_l^1 = \pi/2 - \arctan\left(\frac{2l}{n}\right), \quad \theta_l^2 = \arctan\left(\frac{2l}{n}\right), \tag{8}$$

$k = -n, \ldots, n$, and $l = -n/2, \ldots, n/2$. As we can see in Figure 1(c), for each fixed angle l, the samples of the pp grid are equally spaced along the radial direction. However, this spacing is different for different angles. Also, the grid is not equally spaced in the angle coordinate; it is instead equally spaced in a transformed coordinate: the slope. Formally,

$$\Delta r_k^1 \triangleq r_{k+1}^1 - r_k^1 = \sqrt{4\left(\frac{l}{n}\right)^2 + 1}, \qquad \Delta r_k^2 \triangleq r_{k+1}^2 - r_k^2 = \sqrt{4\left(\frac{l}{n}\right)^2 + 1} \tag{9}$$

and

$$\Delta\theta_{pp}^1(l) \triangleq \cot\theta_{l+1}^1 - \cot\theta_l^1 = \frac{2}{n}, \qquad \Delta\theta_{pp}^2(l) \triangleq \tan\theta_{l+1}^2 - \tan\theta_l^2 = \frac{2}{n}, \tag{10}$$

with r_k^1, r_k^2, θ_l^1, and θ_l^2 given by (7) and (8).

The ppFT is defined as the samples of $\hat{I}$, given by (2), on the pp grid Ω_{pp}, given by (3)–(5). Formally, the ppFT is a linear transformation from $n \times n$ arrays to $2 \times 2n+1 \times n+1$ arrays. Its value $\hat{I}_{\Omega_{pp}^j}(k,l)$ is defined for $j = 1, 2$, $k = -n, \ldots, n$, and $l = -n/2, \ldots, n/2$ by

$$\hat{I}_{\Omega_{pp}^1}(k,l) = \hat{I}\left(-\frac{2l}{n}k, k\right), \tag{11}$$

$$\hat{I}_{\Omega_{pp}^2}(k,l) = \hat{I}\left(k, -\frac{2l}{n}k\right), \tag{12}$$

where $\hat{I}$ is given by (2). We also use operator notation $\mathcal{F}_{pp}I$, meaning

$$(\mathcal{F}_{pp}I)(j,k,l) \triangleq \hat{I}_{\Omega_{pp}^j}(k,l), \tag{13}$$

where $j = 1, 2$, $k = -n, \ldots, n$, and $l = -n/2, \ldots, n/2$.

3. Relation to previous work. A grid much like the pp grid, but coarser, has been proposed several times in the literature under various names. This general type of grid was seemingly first introduced by Mersereau and Oppenheim [29] under the name "concentric-squares grid." Mersereau and Oppenheim worked from the viewpoint of computerized tomography. They assumed that data on a continuum object were gathered in unequally spaced projections chosen so that the 1D Fourier transform corresponded to the concentric-squares grid. They considered the problem of reconstructing a discrete array of n^2 pixels from such Fourier domain data and developed an algorithm based on interpolating from the data given in the concentric-squares grid to the Cartesian grid to approximately reconstruct the values at the Cartesian grid points, followed by standard inverse 2D FFT. In the concentric-squares-to-Cartesian conversion step, Mersereau and Oppenheim used simple 1D interpolation based on linear interpolation in rows/columns.

The difference between [29] and our work is threefold: (i) Mersereau and Oppenheim's grid samples half as frequently as ours in the radial direction. Their concentric-squares grid would result in our framework from making the "expected" choice $m = n$ in (3) in place of our "unconventional" choice $m = 2n+1$. Using $m = 2n+1$ is perhaps not important in the original setting of approximate tomographic reconstruction from coarse noisy data, but it is of crucial importance for our intended applications and interpretation of the transform. Indeed, with the choice $m = 2n + 1$ the transform is related to integration along lines, whereas for $m = n$ it is not. As shown in [3], the original concentric-squares grid does not honor line geometry of the continuous Radon transform, as it involves wraparound of the underlying lines; (ii) Mersereau and Oppenheim's methodology addresses reconstruction (inversion) using data gathered by a medical scanner or other physical integration device; they do not attempt to define a forward transform for digital image data or establish exact invertibility and fast inversion algorithms associated to digital arrays; and (iii) their methodology is approximate; they do not obtain precise evaluation of the transform at points in the concentric-squares grid. Because of the extremely oscillatory nature of the underlying trigonometric polynomial, it is unfortunately not the case that crude interpolations can form the basis of a foundational tool. Closed-form solutions with machine precision ought to be used if they can be found.

Several important later papers in journals devoted to computerized tomography improved on Mersereau and Oppenheim—in both medical tomography [31, 11, 12] and synthetic aperture radar imaging [25]. Like Mersereau and Oppenheim, these authors are concerned with image reconstruction from tomographic data; effectively they assume that one is given data in the Fourier domain on a concentric-squares grid, and the problem is to reconstruct the underlying continuum object.

Pasciak's unpublished work [31], which is known among tomography experts through a citation in Natterer's book [30], showed in 1980 that, given data on a pp grid in Fourier space, one could calculate a collection of n^2 sums which, using the notation of this paper, we can write as

$$\sum c^s_{k,l} e^{i\xi^s_{k,l}(u,v)'}, \quad -n/2 \le u, v < n/2, \tag{14}$$

where the $\xi^s_{k,l}$ are points in the concentric-squares grid. (Pasciak makes no reference to Mersereau and Oppenheim.) In other words, Pasciak showed the way to rapidly and precisely compute the formal adjoint of a ppFT-like transform, which is based on the concentric-squares grid $m = n$ rather than the pp grid $m = 2n + 1$. His paper used the chirp-Z transform to do this in closed form, a clever and fundamental idea which has later been rediscovered or applied several times in this general domain.

Edholm and Herman [11] develop the linogram, which from this paper's viewpoint may be described as follows. Data on a continuum object is gathered at a continuous and complete set of projections, indexed by slope $\tan(\theta)$ rather than angle θ. The mathematical foundations of this continuum transform are developed. In a followup paper with Roberts [12] they consider implementation, and assume a discrete set of projections equispaced in $\tan(\theta)$. By digitally sampling each constant θ projection and taking a 1D DFT of the resulting samples, they argue that they are essentially given data on a concentric-squares grid in Fourier space (making no reference to Mersereau and Oppenheim or to Pasciak). They are concerned with reconstruction, consider the sums of (14), and derive a fast algorithm which is the same as Pasciak's, using again the chirp-Z transform. In their implementation paper [12], Edholm and Herman did in fact suggest the use of $m = 2n+1$ in their discussion of the linogram. However, this arose in discussing the appropriate discretization of the in-principle continuous slope coordinate. The ideas which drive our choice concern the discrete projection-slice theorem and the design of forward and inverse discrete transforms which invert each other. Such issues did not arise in Edholm and Herman's setting. Clearly, Edholm and Herman contributed the fundamental insight that there was a continuum transform that could be discretized compatibly, provided one thinks in terms of slopes rather than angles.

Contemporaneously with Edholm and Herman, Lawton [25] develops a so-called polar Fourier transform for synthetic aperture radar (SAR) imagery. He introduces a concentric-squares grid, assumes that SAR data are essentially given on such a concentric-squares grid in Fourier space, and considers the problem of rapidly reconstructing an image from such data. He considers the sums of (14) and derives a fast algorithm using again the chirp-Z transform. He refers to Mersereau and Oppenheim.

In comparison to our work (i) these works are about reconstruction only, assuming that data are gathered about a continuum object by a physical device, and (ii) the algorithmic problem they consider is equivalent to rapidly computing (14).

The viewpoint in our paper is rather different. We seek to develop a general framework for processing of digital images and solving tasks of representing and manipulating such images; it is most important to us that true lines in image space are faithfully represented by the tools we use.

The tools we have developed for image processing rely on the ppFT as an engine for rapid and precise calculations with digital images; for example, we develop transforms with exact reconstruction properties with this framework [8, 6, 9]. Older ideas would either not give exactness or not correspond to faithful representation of linear features.

In fact, the viewpoint we are developing in this paper has by now proven fruitful in diverse work by several groups [34, 1, 26, 23, 24]. An earlier version of this paper [4] was written in 2000 and was under review for a considerable length of time while groups at Tel Aviv, Yale, Stanford, CalTech, the Technion, CEA Saclay, and Georgia Tech were using these tools to develop image processing applications [8, 6, 9, 2]. This paper extracts content from [4] specifically related to ppFT, refines the earlier content further, and extends it. A significant improvement of this work over [4] is the use of $m = 2n + 1$ ([4] used instead $m = 2n$). The choice $m = 2n + 1$ makes the pp grid centrosymmetric and offers full Hermitian symmetry of the transform.

Software implementations of these ideas are available, for example, at [21]; applications are available at BeamLab [21], as well as [20].

Further differences, related to the discrete Radon transform, are discussed in [3].

4. Fast forward transform. In this section we present a fast algorithm that efficiently computes the ppFT of an image I. The idea is to evaluate the traditional Fourier transform $\hat{I}$ (see (2)) on a Cartesian grid using the 2D FFT algorithm and use an exact interpolation formula to evaluate it on the pp grid. This exact interpolation is efficiently implemented using the *fractional Fourier transform* (frFT).

Consider a vector $c \in \mathbb{C}^{n+1}$ representing a series of equispaced samples along a 1D line. Denote the frFT of c by

$$(15) \qquad (F^{\alpha}_{n+1}c)(k) = \sum_{u=-n/2}^{n/2} c(u)e^{-2\pi i\alpha ku/(n+1)}, \quad k = -n/2, \dots, n/2, \quad \alpha \in \mathbb{R}.$$

An important property of the frFT is that, given a vector c of length $n + 1$, the sequence $(F^{\alpha}_{n+1}c)(k)$, $k = -n/2, \dots, n/2$, can be computed using $O(n \log n)$ operations for any $\alpha \in \mathbb{R}$ (see [5]). Equation (15) is usually referred to as the unaliased frFT, and it differs from the usual definition of the frFT given in [5]. The algorithm that computes the unaliased frFT (15) is very similar to the algorithm in [5] and is therefore omitted. MATLAB and C implementations are freely available [21].

Our ppFT algorithm pipelines several operators:

- $E_{m,n}$: Padding operator. $E_{m,n}I$ symmetrically zero pads an image I of size $n \times n$ to size $m \times n$.
- F_1^{-1}: 1D inverse discrete Fourier transform (DFT).
- $\tilde{F}^{\alpha}_m$: frFT with factor α. The operator takes a sequence of length n, symmetrically zero pads it to length $m = 2n+1$, applies to it the frFT F^{α}_m with scale factor α, and returns the $n+1$ central elements.
- F_2: 2D DFT.
- $G_{k,n}$: Resampling operator given by

$$G_{k,n} = \tilde{F}^{\alpha}_m \circ F_1^{-1}, \quad \alpha = 2k/n. \tag{16}$$

Using this notation, Algorithm 1 displayed below computes the Ω^1_{pp} sector of the ppFT $\hat{I}_{\Omega^1_{pp}}(k,l)$ (see (11)). To compute the Ω^2_{pp} sector $\hat{I}_{\Omega^2_{pp}}(k,l)$ (see (12)) simply switch the roles of the x and y axes in Algorithm 1.

Algorithm 1. Computing the ppFT $\hat{I}_{\Omega^1_{pp}}$

Input: Image I of size $n \times n$

Output: Array Res_1 with $n+1$ rows and $m = 2n+1$ columns that contains the samples of $\hat{I}_{\Omega^1_{pp}}$

```
1: $m \leftarrow 2n+1$
2: $\hat{I}_d \leftarrow F_2 \circ E_{m,n} I$
3: for $k = -n, \dots, n$ do
4:     $q \leftarrow \hat{I}_d(\cdot, k)$
5:     $w_k \leftarrow G_{k,n}(q)$,        $w_k \in \mathbb{C}^{n+1}$
6:     $Res_1(k,l) \leftarrow w_k(-l)$
7: end for
```

We now show that Algorithm 1 computes the ppFT.

THEOREM 4.1 (correctness of Algorithm 1). *Upon termination of Algorithm* 1 *we have*

$$Res_1(k,l) = \hat{I}_{\Omega^1_{pp}}(k,l), \tag{17}$$

where $k = -n, \dots, n$, $l = -n/2, \dots, n/2$, and $\hat{I}_{\Omega^1_{pp}}$ is given by (11).

Proof. After completion of step 2 in Algorithm 1, the (l,k) element of $\hat{I}_d$ is given by

$$\hat{I}_d(l,k) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u,v) e^{-2\pi i l u/n} e^{-2\pi i k v/m}, \tag{18}$$

where $l = -n/2, \dots, n/2-1$, $k = -n, \dots, n$, and $m = 2n+1$. Turn to calculation of $Res_1(k_0, j)$ for some fixed k_0. Take row k_0 from $\hat{I}_d$, and denote the resulting vector of length n by q

$$q(l) = \hat{I}_d(l, k_0), \qquad l = -n/2, \dots, n/2-1. \tag{19}$$

Step 5 in Algorithm 1 defines $w_k(j) = (G_{k_0,n}(q))_j$, where, according to (16), $G_{k_0,n}(q) = \tilde{F}_m^{2k_0/n}(F_1^{-1}(q))$. We begin by evaluating $F_1^{-1}(q)$. By expanding (19) using (18) we get

$$(20)\qquad q(l) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u,v) e^{-2\pi i l u/n} e^{-2\pi i k_0 v/m} = \sum_{u=-n/2}^{n/2-1} c_{k_0}(u) e^{-2\pi i l u/n},$$

where

$$(21)\qquad c_{k_0}(u) = \sum_{v=-n/2}^{n/2-1} I(u,v) e^{-2\pi i k_0 v/m}, \quad u = -n/2, \dots, n/2-1.$$

Equation (20) states that the vector q is the DFT of $\{c_{k_0}(u)\}$. Therefore, $F_1^{-1}(q) = \{c_{k_0}(u)\}$. Finally, taking the frFT $\tilde{F}_m^{2k_0/n}$ of the array $\{c_{k_0}(u)\}$ using (15) and (21) gives $w_{k_0}(j) = (\tilde{F}_m^{2k_0/n}(\{c_{k_0}(u)\}))_j = \hat{I}(2jk_0/n, k_0)$, from which we conclude that upon completion of step 6 in Algorithm 1

$$Res_1(k_0, j) = w_{k_0}(-j) = \hat{I}(-2jk_0/n, k_0) = \hat{I}_{\Omega^1_{pp}}(k_0, j), \quad j = -n/2, \dots, n/2. \qquad \square$$

Next, we analyze the complexity of Algorithm 1. Step 2 can be implemented in $O(n^2 \log n)$ operations by using successive applications of 1D FFT. Each call to $G_{k,n}$ in step 5 involves the application of a 1D inverse Fourier transform ($O(n \log n)$ operations) followed by the computation of an frFT ($O(n \log n)$ operations) and thus requires $O(n \log n)$ operations. Step 5 computes $G_{k,n}$ for each row k ($2n+1$ rows), which requires a total of $O(n^2 \log n)$ operations. Step 6 involves flipping $2n+1$ vectors of length $n+1$, which requires a total of $O(n^2)$ operations. Thus, the total complexity of Algorithm 1 is $O(n^2 \log n)$ operations.

With an optimized implementation, the complexity of computing the ppFT of an $n \times n$ image is $100n^2 \log_2 n$ operations. Note that the number of frequency samples in the output array is roughly $4n^2$. Computing $4n^2$ Cartesian frequency samples using the 2D FFT requires $20n^2 \log_2 n$ operations. Thus, computing the ppFT is only 5 times slower than the 2D FFT. This complexity analysis assumes that the 1D FFT of a vector of length n requires $5n \log_2 n$ operations, and that the frFT of a vector of length n can be computed in $20n \log_2 n$ operations (independent of α) [5].

Algorithm 1 suggests a way to rapidly compute the adjoint ppFT in $O(n^2 \log n)$. In effect, the algorithm represents the output as a result of pipelining several linear operators. Since the adjoint of a composition is the composition of adjoints in reversed order, we can compute the adjoint ppFT by reversing the order of execution in Algorithm 1 and replacing each line by its adjoint. The resulting algorithm, given in Algorithm 2, uses the following operators:

- U_n: Truncation operator. The operator $U_n I$ takes an image I and returns its $n \times n$ central elements.
- F_2^{-1}: 2D inverse DFT.
- adj $G_{k,n}$: Adjoint of the operator $G_{k,n}$ (see (16)); formally

$$(22)\qquad \text{adj } G_{k,n} = \frac{1}{n} F_1 \circ \text{adj } \tilde{F}_m^{\alpha}, \quad \alpha = 2k/n,$$

where F_1 is the 1D DFT and adj $\tilde{F}_m^{\alpha}$ is an operator that takes a vector of length $n+1$, symmetrically zero pads it to length $m = 2n+1$, applies the frFT with factor $-\alpha$, and returns the n central elements.

Algorithm 2. Fast adjoint ppFT

```
1: for $k = -n, \ldots, n$ do
2:     $q(l) \leftarrow \hat{I}_{\Omega^1_{pp}}(k, -l), \quad l = -n/2, \ldots, n/2$
3:     $\tilde{I}_1(k, \cdot) \leftarrow (\text{adj } G_{k,n})q$
4: end for
5: $\tilde{I}_1 \leftarrow U_n(mnF_2^{-1}(\tilde{I}_1))$
6: for $k = -n, \ldots, n$ do
7:     $q(l) \leftarrow \hat{I}_{\Omega^2_{pp}}(k, -l), \quad l = -n/2, \ldots, n/2$
8:     $\tilde{I}_2(\cdot, k) \leftarrow (\text{adj } G_{k,n})q$
9: end for
10: $\tilde{I}_2 \leftarrow U_n(mnF_2^{-1}(\tilde{I}_2))$
11: $\tilde{I} \leftarrow \tilde{I}_1 + \tilde{I}_2$
```

5. Invertibility.

Suppose we are given the values of the ppFT $\mathcal{F}_{pp}I$. It is possible to recover I, as we now show. Consider a vector of samples from $\hat{I}_{\Omega^1_{pp}}$ that corresponds to some $k_0 \neq 0$. From (11)

$$\hat{I}_{\Omega^1_{pp}}(k_0, j) = \hat{I}(-2jk_0/n, k_0), \quad j = -n/2, \ldots, n/2.$$

From (2)

$$\hat{I}(-2jk_0/n, k_0) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i(-2k_0/n)ju/m} e^{-2\pi i k_0 v/m}, \tag{23}$$

which we write as

$$\hat{I}(-2jk_0/n, k_0) = \sum_{u=-n/2}^{n/2-1} c_{k_0}(u) e^{-2\pi i(-2jk_0/n)u/m}, \tag{24}$$

where

$$c_{k_0}(u) = \sum_{v=-n/2}^{n/2-1} I(u, v) e^{-2\pi i k_0 v/m}, \quad u = -n/2, \ldots, n/2 - 1.$$

Denote $T_{k_0}(-2jk_0/n) \triangleq \hat{I}(-2jk_0/n, k_0)$. $T_{k_0}(-2jk_0/n)$ are the values of the trigonometric polynomial

$$T_{k_0}(x) = \sum_{u=-n/2}^{n/2-1} c_{k_0}(u) e^{-2\pi i x u/m} \tag{25}$$

at the points $\{-2jk_0/n\}$, $j = -n/2, \ldots, n/2$. Since $k_0 \neq 0$ we have the values of $T_{k_0}(x)$ at $n + 1$ distinct points $\{-2jk_0/n\}$. Therefore, we can uniquely determine $\{c_{k_0}(u)\}$ and $T_{k_0}(x)$. By evaluating $T_{k_0}(x)$ at integer points using (25), (24), and (2), we get $T_{k_0}(j) = \hat{I}(j, k_0)$, which means that we can recover the DFT of I for all $k_0 \neq 0$. Therefore, it remains to recover $\hat{I}(j, 0)$, $j = -n/2, \ldots, n/2$.

By taking a sequence of samples from $\hat{I}_{\Omega_{pp}^2}$ (see (12)), which corresponds to some $k_0 \neq 0$, we get $\hat{I}_{\Omega_{pp}^2}(k_0, j) = \hat{I}(k_0, -2jk_0/n)$, and using (2) we write

$$T'_{k_0}(-2jk_0/n) \triangleq \hat{I}(k_0, -2jk_0/n) = \sum_{v=-n/2}^{n/2-1} c'_{k_0}(v) e^{-2\pi i(-2jk_0/n)v/m},$$

where

$$c'_{k_0}(v) = \sum_{u=-n/2}^{n/2-1} I(u,v) e^{-2\pi i k_0 u/m}, \quad v = -n/2, \ldots, n/2-1. \tag{26}$$

$\{T'_{k_0}(-2jk_0/n)\}$ are the values of the trigonometric polynomial

$$T'_{k_0}(x) = \sum_{v=-n/2}^{n/2-1} c'_{k_0}(v) e^{-2\pi i x v/m}$$

at $n+1$ distinct points $\{-2jk_0/n\}$, $j = -n/2, \ldots, n/2$, and thus uniquely determine $\{c'_{k_0}(v)\}$ and $T'_{k_0}(x)$. By evaluating $T'_{k_0}(x)$ at integer points and using (26) and (2), we get $T'_{k_0}(j) = \hat{I}(k_0, j)$ for $j = -n/2, \ldots, n/2$. Specifically, we can evaluate $\hat{I}(k_0, 0)$ for $k_0 \neq 0$, which means that we can recover $\hat{I}$ at all Cartesian grid points except the origin. Since at the origin $\hat{I}_{\Omega_{pp}^1}(0,0) = \hat{I}(0,0)$, we have the values of $\hat{I}(\xi_1, \xi_2)$ on the entire discrete Cartesian grid; that is, we can recover the DFT of I from $\hat{I}_{\Omega_{pp}}$. Finally, we can recover I by using the 2D inverse DFT. Hence, the 2D ppFT is invertible.

6. Iterative inverse algorithm.

We are given a vector y which purports to be the transform $\mathcal{F}_{pp}x$ of a vector x, and we are asked to recover x. Since y is not necessarily in the range of the ppFT, e.g., because of noise or measurement errors, we actually solve

$$\min_{x \in \mathcal{D}(\mathcal{F}_{pp})} \|\mathcal{F}_{pp}x - y\|_2 \tag{27}$$

instead, where $\mathcal{D}(\mathcal{F}_{pp})$ is the domain of the ppFT. Solving (27) is equivalent to solving the normal equations

$$\mathcal{F}_{pp}^* \mathcal{F}_{pp} x = \mathcal{F}_{pp}^* y, \tag{28}$$

where $\mathcal{F}_{pp}^*$ is the adjoint ppFT. Since $\mathcal{F}_{pp}^*\mathcal{F}_{pp}$ is symmetric and positive definite, we can use the conjugate-gradient method [17] to solve (28). When using the conjugate-gradient method, we never explicitly form the matrices that correspond to $\mathcal{F}_{pp}$ and $\mathcal{F}_{pp}^*$ (which are huge), since only their applications to a vector are required. As shown in section 4, both the ppFT and its adjoint can be applied in $O(n^2 \log n)$ operations. Moreover, very little extra storage is required by the conjugate-gradient algorithm (as opposed to other iterative schemes such as, for example, GMRES), so that the method can be used to solve very large problems. The initial guess used for the conjugate-gradient method is zero. As we see below, we get excellent convergence even with this trivial initial guess.

The number of iterations required by the conjugate-gradient method depends on the condition number of the transform. To accelerate convergence, we apply the method of preconditioners [18], replacing the normal equations (28) by the rescaled system

$$\mathcal{F}_{pp}^{*} M \mathcal{F}_{pp} x = \mathcal{F}_{pp}^{*} M y. \tag{29}$$

Here M is a diagonal scaling matrix chosen, so the condition number of $\mathcal{F}_{pp}^{*} M \mathcal{F}_{pp}$ is much smaller than the condition number of $\mathcal{F}_{pp}^{*}\mathcal{F}_{pp}$ or such that the eigenvalues of $\mathcal{F}_{pp}^{*} M \mathcal{F}_{pp}$ are well clustered.

In our case, we use the diagonal preconditioner M defined by (see (13))

$$(My)(s,k,l) = w_{k,l}\, y(s,k,l), \qquad w_{k,l} = \begin{cases} \frac{1}{m^2}, & k = 0, \\ \frac{2(n+1)|k|}{nm} & \text{otherwise}, \end{cases} \tag{30}$$

where $s = 1,2$, $k = -n,\dots,n$, $l = -n/2,\dots,n/2$, and $m = 2n+1$. This weighting can be understood as an example of the principle of *density compensation.* Roughly speaking, each square of size $1/n$ by $1/n$ is responsible for an equal part of the overall ℓ_2 norm. However the pp grid samples certain squares much more finely than others. By applying the weighting just described, we make sure that the weighted samples falling inside a given cell account for the correct fraction of the ℓ_2 norm. We mention that the paper [13] proposes a method for designing effective preconditioners (weights) in the 1D case. The performance of these preconditioners can be guaranteed in terms of the properties of the sampling points. Unfortunately, the arguments in [13] apply only to the 1D case, and the construction therein cannot be applied to our case.

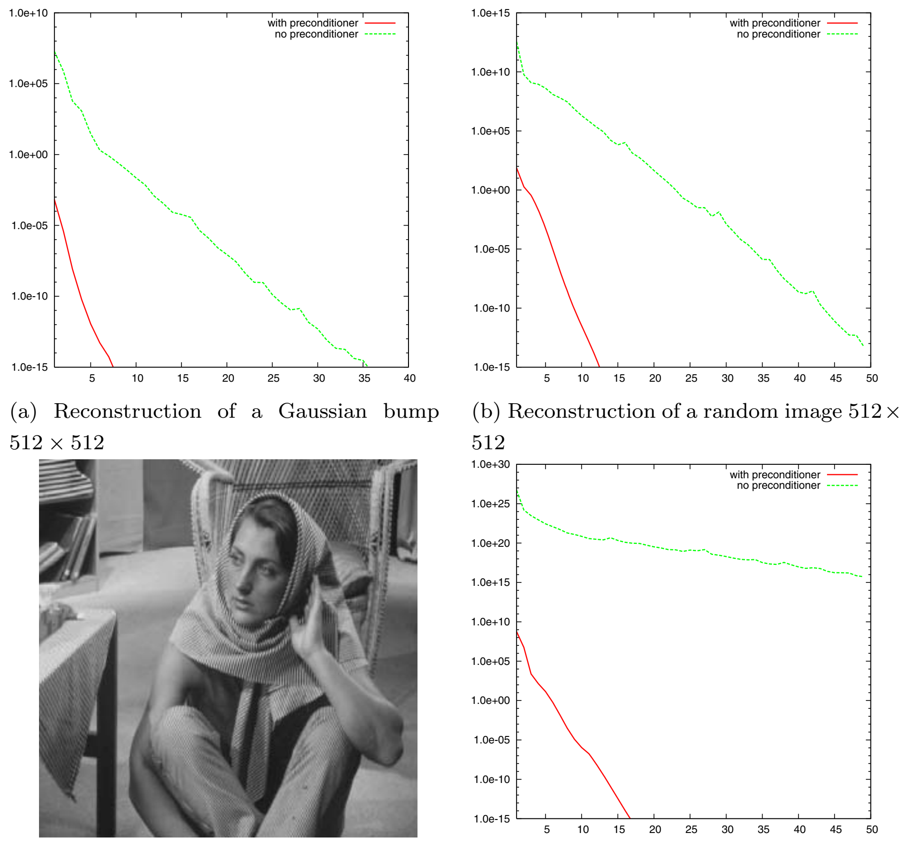
The efficiency of our preconditioner M (see (30)) is demonstrated in Figure 2. Each graph presents the residual error of the conjugate-gradient iteration as a function of the iteration number. In Figure 2(a), the original image is a 2D Gaussian bump of size 512×512 with $\mu_x = \mu_y = 0$ and $\sigma_x = \sigma_y = \frac{512}{6}$. In Figure 2(b), the original image is a random image of size 512×512, whose entries are uniformly distributed between 0 and 1. In Figure 2(d), the original image is Barbara of size 512×512, shown in Figure 2(c). As we can see from Figures 2(a)–(d), our preconditioner significantly accelerates convergence. With the preconditioner, only a few iterations are required, and the number of iterations is nearly independent of the content of the reconstructed image.

Tables 1 and 2 quantify the performance of the iterative inversion algorithm for images of various sizes. Table 1 presents the inversion of the ppFT of a Gaussian bump. Table 2 presents the inversion of the ppFT of a random image, whose entries are uniformly distributed between 0 and 1. In both tables, the error tolerance of the conjugate-gradient method is set to $\varepsilon = 10^{-12}$. The tables were generated as follows. Given an image I (random or Gaussian bump), its ppFT is computed. Then the iterative inversion algorithm is applied to recover the image. We denote by $\tilde{I}$ the reconstructed image. We evaluate reconstruction quality with these error measures:

$$E_2 = \frac{\sqrt{\sum_{u,v} \left|\tilde{I}(u,v) - I(u,v)\right|^2}}{\sqrt{\sum_{u,v} |I(u,v)|^2}}, \qquad E_\infty = \frac{\max_{u,v} \left|\tilde{I}(u,v) - I(u,v)\right|}{\max_{u,v} |I(u,v)|}, \tag{31}$$

where I is the original image.

Tables 1 and 2 show that very few iterations are required to invert the ppFT with high accuracy. The total complexity of the iterative inversion of the ppFT is



(a) Reconstruction of a Gaussian bump 512×512

(b) Reconstruction of a random image 512×512

(c) Barbara 512×512

(d) Reconstruction of Barbara 512×512

FIG. 2. *The effect of using the preconditioner given in* (30).

TABLE 1

Iterative inversion of the ppFT of a Gaussian bump. Column 1*: n, size of the original $n \times n$ image. Column* 2*: r, residual error of the conjugate-gradient algorithm. Column* 3*: E_2, relative ℓ_2 reconstruction error. Column* 4*: E_∞, relative maximum reconstruction error. Column* 5*: iter, the number of iterations until termination condition. Column* 6*: t, running time in seconds.*

n	r	E_2	E_∞	iter	t (sec)
8	2.80682e-014	2.47277e-007	1.60617e-007	9	0.359
16	1.14334e-013	4.92517e-007	3.86542e-007	8	0.563
32	5.99921e-014	3.44244e-007	2.92515e-007	8	1.202
64	1.05319e-013	4.67737e-007	5.92969e-007	7	2.625
128	6.05610e-013	1.16930e-006	2.56236e-006	6	7.295
256	1.09915e-013	4.94793e-007	1.60205e-006	6	26.789
512	4.30207e-013	9.87174e-007	5.05849e-006	5	83.525
1024	7.02642e-014	4.16717e-007	3.00086e-006	5	373.172

$O(\iota(\varepsilon)n^2 \log n)$, where $\iota(\varepsilon)$ is the number of iterations required to achieve accuracy ε. As we can see from Table 2, the value of $\iota(\varepsilon)$ depends very weakly on the size of the reconstructed image, and in any case $\iota(10^{-12}) \leq 10$.

7. Direct inverse algorithm. A charming feature of the iterative inversion algorithm is the elegant deployment of general principles; but charm and elegance come at a price. First, since the iterative inversion is based on a generic linear algebra

Table 2
Iterative inversion of the ppFT of a random image. For the legend, see Table 1.

n	r	E_2	E_∞	iter	t (sec)
8	4.56049e-014	3.33796e-007	5.21815e-007	9	0.391
16	2.21072e-013	7.13164e-007	1.06025e-006	9	0.672
32	6.30229e-013	1.27807e-006	3.81621e-006	9	1.359
64	3.72178e-013	9.30674e-007	4.31200e-006	9	3.250
128	1.40414e-013	5.43102e-007	2.27508e-006	10	11.562
256	1.69596e-013	5.82115e-007	1.95609e-006	10	43.484
512	1.25562e-013	5.05263e-007	2.47555e-006	10	158.641
1024	8.84166e-014	4.49097e-007	3.73745e-006	10	688.937

approach, it does not utilize the special frequency domain structure of the transform. Second, while the iterative inversion algorithm is shown to have an acceptable empirical convergence rate, the exact number of iterations and thus its running time depend on the specific image to invert. See, for example, the different number of iterations required for a Gaussian bump and a random image in Tables 1 and 2. Third, the conjugate-gradient method enables one to estimate the reconstruction error in terms of the residual error. This error is related to the actual reconstruction error through the norm of the operator, which is difficult to estimate.

We now develop a direct inversion algorithm overcoming these limitations. It is tailored to the specific structure of the given transform. Its running time is independent of the specific image to invert. Also, its accuracy can be estimated in terms of the actual reconstruction error.

Our direct inversion algorithm consists of two phases. The first phase resamples the ppFT into a Cartesian frequency grid; the second phase recovers the image from these Cartesian frequency samples. Resampling from the pseudo-polar to a Cartesian frequency grid is based on an "onion-peeling" procedure, which recovers a single row/column of the Cartesian grid in each iteration, from the outermost row/column to the origin. Recovering each row/column is based on a fast algorithm that resamples trigonometric polynomials from one set of frequencies to another set of frequencies. This algorithm is approximate but arbitrarily accurate; its running time depends logarithmically on the required accuracy ε.

For a different approach to the inversion of the concentric-squares grid, which is based on 1D nonequally spaced FFTs; see [32]. The approach therein is based on a grid with $m = n$ and not $m = 2n + 1$ as in our case. The fundamental nature of this difference was discussed in section 3 above.

This section is organized as follows. In section 7.1 we present the mathematical tools used by the algorithm. Specifically, we present a fast algorithm that, for a given Toeplitz matrix A_n of size $n \times n$, applies A_n^{-1} to an arbitrary vector in $O(n \log n)$ operations. In section 7.2 we present the outline of the algorithm that inverts the ppFT. Section 7.3 describes the operators that resample from the pseudo-polar to a Cartesian frequency grid. Section 7.4 describes the procedure that recovers the image from this Cartesian frequency grid. Finally, section 7.5 provides numerical examples that demonstrate the accuracy and efficiency of the proposed algorithm.

7.1. Solving Toeplitz systems. Let A_n be a Toeplitz matrix of size $n \times n$, and let y be an arbitrary vector of length n. We now describe a fast algorithm to compute $A_n^{-1}y$. The algorithm consists of fast factorization of the inverse Toeplitz matrix, followed by a fast algorithm that applies the inverse matrix on a vector. The approach—circulant embedding—is well known [15, 22]; we include it for the sake of completeness. Let $T_n(c,r)$ denote an $n \times n$ Toeplitz matrix whose first column and row are c and r, respectively.

Circulant matrices are diagonalized by the Fourier matrix; hence the circulant matrix C_n can be written $C_n = W_n^* D_n W_n$, where D_n is a diagonal matrix containing the eigenvalues $\lambda_1, \ldots, \lambda_n$ of C_n and W_n is the Fourier matrix, given by $W_n(j,k) = \frac{1}{\sqrt{n}} e^{2\pi ijk/n}$. Moreover, if $c = [c_0, c_1, \ldots, c_{n-1}]^T$ is the first column of C_n, then $W_n c = [\lambda_1, \ldots, \lambda_n]^T$. Obviously, the matrices W_n and W_n^* can be applied in $O(n \log n)$ operations; this is simply the FFT. The multiplication of C_n with an arbitrary vector x of length n can be implemented in $O(n \log n)$ operations by applying a FFT to x, multiplying the result by D_n, and taking the inverse FFT.

To compute $A_n x$ for an arbitrary Toeplitz matrix $A_n = T_n(c,r)$ and an arbitrary vector x, we first embed A_n in a circulant matrix C_{2n} of size $2n \times 2n$

$$C_{2n} = \begin{pmatrix} A_n & B_n \\ B_n & A_n \end{pmatrix},$$

where B_n is a $n \times n$ Toeplitz matrix given by

$$B_n = T_n([0, r_{n-1}, \ldots, r_2, r_1], [0, c_{n-1}, \ldots, c_2, c_1]).$$

Then $A_n x$ is computed in $O(n \log n)$ operations by zero padding x to length $2n$, applying C_{2n} to the padded vector, and discarding the last n elements of the result vector.

Next, assume that A_n is invertible. The Gohberg–Semencul formula [15, 16] provides a representation of A_n^{-1} as

$$A_n^{-1} = \frac{1}{x_0}\left(M_1 M_2 - M_3 M_4\right), \tag{32}$$

where

$$M_1 = T_n([x_0, x_1, \ldots, x_{n-1}], [x_0, 0, \ldots, 0]),$$

$$M_2 = T_n([y_{n-1}, 0, \ldots, 0], [y_{n-1}, y_{n-2}, \ldots, y_0]),$$

$$M_3 = T_n([0, y_0, \ldots, y_{n-2}], [0, \ldots, 0]),$$

$$M_4 = T_n([0, \ldots, 0], [0, x_{n-1}, \ldots, x_1]),$$

$x = [x_0, \ldots, x_{n-1}]$ is the solution of $A_n x = e_0$, $y = [y_0, \ldots, y_{n-1}]$ is the solution of $A_n y = e_{n-1}$, $e_0 = [1, 0, \ldots, 0]^T$, and $e_{n-1} = [0, \ldots, 0, 1]^T$. The matrices M_1, M_2, M_3, and M_4 have Toeplitz structure and are represented implicitly using the vectors x and y. Hence, the total storage required to store M_1, M_2, M_3, and M_4 is $2n$ elements. If the matrix A_n is fixed, then the vectors x and y can be precomputed. Once the triangular Toeplitz matrices M_1, M_2, M_3, M_4 are computed, the application of A_n^{-1} is reduced to the application of four Toeplitz matrices, and thus the application of A_n^{-1} to a vector requires $O(n \log n)$ operations.

7.2. Outline of the direct inversion algorithm. For an image I of size $n \times n$, we define the array $\hat{I}_D$ to be

$$\hat{I}_D(k,l) = \sum_{u,v=-n/2}^{n/2-1} I(u,v) e^{-2\pi i(2ku+2lv)/m}, \quad k,l = -n/2, \ldots, n/2, \tag{33}$$

where $m = 2n + 1$. $\hat{I}_D(k, l)$ is obtained from the image I by symmetrically zero padding it to size $(2n + 1) \times (2n + 1)$, applying the 2D FFT on the padded image, and discarding every other sample along each dimension.

Our algorithm for inverting the ppFT has two phases. The first computes the array $\hat{I}_D$ from the samples of the ppFT. The second recovers the image I from the array $\hat{I}_D$. The first phase processes each row/column of the pp grid, from the outermost rows/columns to the origin, where at step i of this phase ($i = 0, \dots, n/2$), it recovers rows/columns $i - n/2$ and $-(i - n/2)$ of $\hat{I}_D$ from rows/columns $2(i - n/2)$ and $-2(i - n/2)$ of the pp grid. This is depicted in Figure 3. Light circles represent samples of the pp grid. Dark circles represent samples of $\hat{I}_D$. The outermost rows and columns of $\hat{I}_D$ are simply the outermost rows/columns of $\hat{I}_{\Omega^1_{pp}}$ and $\hat{I}_{\Omega^2_{pp}}$ ((11) and (12)), respectively (Figures 3(a) and 3(b)). Rows $-n/2+1$ and $n/2-1$ of $\hat{I}_D$ are recovered from rows $-n+2$ and $n-2$ of $\hat{I}_{\Omega^1_{pp}}$ and from the columns of $\hat{I}_D$ recovered in step 1 (Figure 3(c)). Similarly, columns $-n/2+1$ and $n/2-1$ of $\hat{I}_D$ are recovered from columns $-n+2$ and $n-2$ of $\hat{I}_{\Omega^2_{pp}}$ and from the rows of $\hat{I}_D$ recovered in step 1 (Figure 3(d)). Rows $-n/2+2$ and $n/2-2$ of $\hat{I}_D$ are recovered from rows $-n+4$ and $n-4$ of $\hat{I}_{\Omega^1_{pp}}$ and from the columns of $\hat{I}_D$ recovered in steps 1 and 2. Columns $-n/2+2$ and $n/2-2$ of $\hat{I}_D$ are recovered from columns $-n+4$ and $n-4$ of $\hat{I}_{\Omega^2_{pp}}$ and from the rows of $\hat{I}_D$ recovered in steps 1 and 2. These steps continue until all of the samples of $\hat{I}_D$ are recovered. As we see from Figure 3, the Cartesian grid samples $\hat{I}_D$ are recovered from the outside to the origin row by row and column by column. For this reason we refer to this procedure as onion-peeling inversion. At each step, we recover the next row/column of the Cartesian grid by using the samples of the corresponding row/column in the pp grid and the columns/rows of $\hat{I}_D$ recovered in previous steps. Since the radial resolution of the pp grid is $2n + 1$, only half of the rows/columns of the pp grid are used to recover $\hat{I}_D$.

Let $H^h_{n,k}$ denote the operator recovering row k ($k = -n/2, \dots, n/2$) of $\hat{I}_D$ from row $2k$ of $\hat{I}_{\Omega^1_{pp}}$ and from columns $|j| > |k|$ of $\hat{I}_D$. Similarly, let $H^v_{n,k}$ denote the operator recovering column k of $\hat{I}_D$ from column $2k$ of $\hat{I}_{\Omega^2_{pp}}$ and from rows $|j| > |k|$ of $\hat{I}_D$. As depicted in Figure 3, $H^h_{n,k}$ and $H^v_{n,k}$ operate on vectors of even length and return vectors of even length (it is easier to implement them for even lengths). Also, $H^h_{n,k}$ and $H^v_{n,k}$ always return vectors of length n, although some of the returned values were already computed in previous steps. This simplifies implementation while not worsening the order of computational complexity of the scheme. In section 7.3 we give a formal description of $H^h_{n,k}$ and $H^v_{n,k}$ and describe a fast algorithm that implements them to any prescribed accuracy ε. Then in section 7.4 we present an algorithm that recovers I from $\hat{I}_D$ (see (33)).

Algorithm 3 provides the pseudocode for the direct inversion algorithm. Each application of the operators $H^h_{n,k}$ and $H^v_{n,k}$ requires $O(n \log n + n \log (1/\varepsilon))$ operations, where ε is the prescribed accuracy of the reconstruction. Hence, the loop in lines 2–7 of Algorithm 3 recovers $\hat{I}_D$ to accuracy ε using $O(n^2 \log n + n^2 \log (1/\varepsilon))$ operations. Recovering I from $\hat{I}_D$ requires $O(n^2 \log n)$ operations. Hence, the total complexity of the direct inversion algorithm is $O(n^2 \log n + n^2 \log (1/\varepsilon))$ operations, where ε is the required accuracy.

7.3. Operators $H^h_{n,k}$ and $H^v_{n,k}$. In this section we provide a detailed description of the operator $H^h_{n,k}$, as well as an efficient algorithm to apply it. This algorithm is approximate and has computational complexity $O(n \log n + n \log (1/\varepsilon))$, with ε

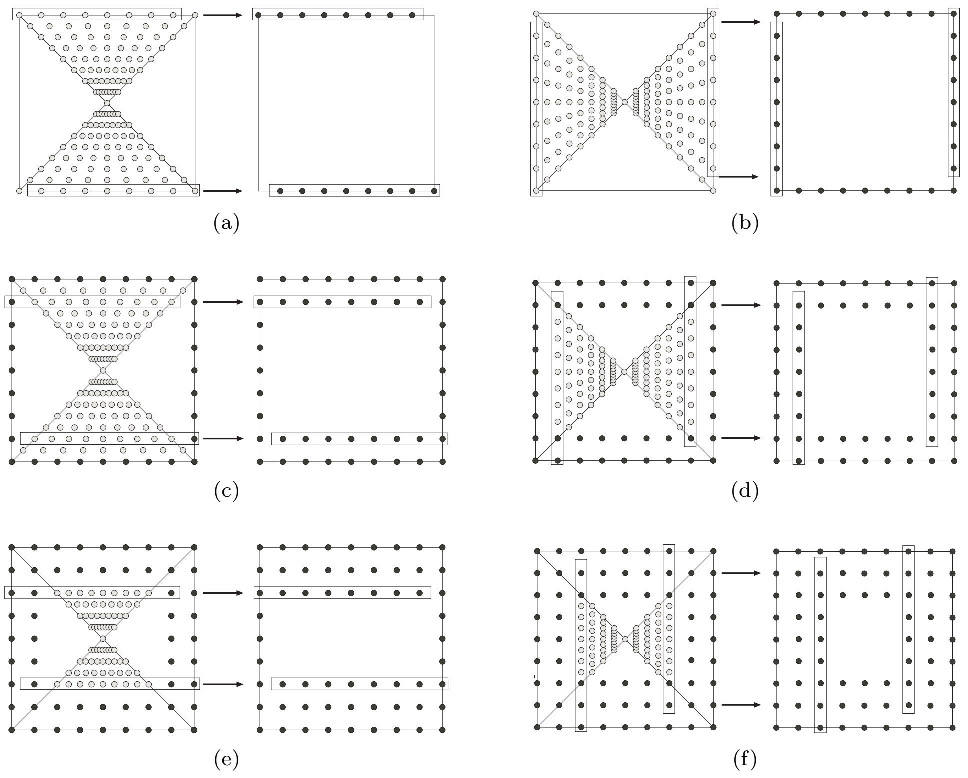


FIG. 3. *Outline of the onion-peeling algorithm for inverting the ppFT.*

Algorithm 3. Inversion of ppFT by onion-peeling

```
Input: ppFT $\hat{I}_{\Omega^1_{pp}}$ and $\hat{I}_{\Omega^2_{pp}}$ (see (11) and (12))
Output: Image $I$ of size $n \times n$
1: $\hat{I}_D \leftarrow \text{zeros}(n+1, n+1)$
2: for $k = -n/2, \ldots, 0$ do
3:     $\hat{I}_D(k,:) \leftarrow H^h_{n,k}(\hat{I}_{\Omega^1_{pp}}, \hat{I}_D)$
4:     $\hat{I}_D(-k,:) \leftarrow H^h_{n,-k}(\hat{I}_{\Omega^1_{pp}}, \hat{I}_D)$
5:     $\hat{I}_D(:,k) \leftarrow H^v_{n,k}(\hat{I}_{\Omega^2_{pp}}, \hat{I}_D)$
6:     $\hat{I}_D(:,-k) \leftarrow H^v_{n,-k}(\hat{I}_{\Omega^2_{pp}}, \hat{I}_D)$
7: end for
8: Recover $I$ from $\hat{I}_D$
```

being the accuracy of the computation. The construction of $H^v_{n,k}$ is similar.

Let $\Omega_{n,k}$ denote samples of the pp grid Ω^1_{pp} (see (4)) belonging to row k:

$$\Omega_{n,k} \triangleq \left\{ \left(-\frac{2l}{n}k, k \right), \ l = -n/2, \ldots, n/2 \right\}.$$

Let

$$\Lambda_{n,k} \triangleq \Omega_{n,2k} \cup C_{n,k}, \tag{34}$$

where

$$C_{n,k} \triangleq \begin{cases} C^1_{n,k}, & k = -n/2, \ldots, 0, \\ C^2_{n,k}, & k = 1, \ldots, n/2, \end{cases} \tag{35}$$

$$C^1_{n,k} = \left\{ (2j, 2k) \;\mid j = -\frac{n}{2}, \ldots, -|k|-1, |k|+1, \ldots, \frac{n}{2}-1 \right\}, \tag{36}$$

$$C^2_{n,k} = \left\{ (2j, 2k) \;\mid j = -\frac{n}{2}+1, \ldots, -|k|-1, |k|+1, \ldots, \frac{n}{2} \right\}. \tag{37}$$

For $k = -n/2$ or $k = n/2$ we have $C_{n,k} = \emptyset$. The set $\Lambda_{n,k}$, given by (34), contains samples of two densities: $n - 2|k| - 1$ samples with spacing 2 and $n+1$ samples with spacing $-2k/n$.

Define the set $\tilde{C}_{n,k}$ as

$$\tilde{C}_{n,k} \triangleq \begin{cases} \tilde{C}^1_{n,k}, & k = -n/2, \ldots, 0, \\ \tilde{C}^2_{n,k}, & k = 1, \ldots, n/2, \end{cases} \tag{38}$$

where

$$\tilde{C}^1_{n,k} = \left\{ (2j, 2k) \;\mid j = -\frac{n}{2}, \ldots, \frac{n}{2}-1 \right\}, \tag{39}$$

$$\tilde{C}^2_{n,k} = \left\{ (2j, 2k) \;\mid j = -\frac{n}{2}+1, \ldots, \frac{n}{2} \right\}. \tag{40}$$

The operator $H^h_{n,k}$ takes the values $\hat{I}_{\Lambda_{n,k}}$, which are the values of $\hat{I}$ (see (2)) on the set $\Lambda_{n,k}$ (from (34)), and evaluates the values of $\hat{I}$ on the set $\tilde{C}_{n,k}$. In other words, the operator $H^h_{n,k}$ resamples the trigonometric polynomial $\hat{I}$ from the set $\Lambda_{n,k}$ to the set $\tilde{C}_{n,k}$. For a fixed k, the samples of $\hat{I}$ on the set $\Lambda_{n,k}$ can be written as the samples of some univariate trigonometric polynomial. Hence, for a fixed k, if we consider $\Lambda_{n,k}$ and $\tilde{C}_{n,k}$ as 1D sets, then the operator $H^h_{n,k}$ resamples a univariate trigonometric polynomial from the set $\Lambda_{n,k}$ to the set $\tilde{C}_{n,k}$. Thus, we implement the operator $H^h_{n,k}$ as follows. Let k be a fixed integer in the range $-n/2, \ldots, 0$ (the construction for $k = 1, \ldots, n/2$ is similar). We choose for each point $p_j \in \tilde{C}_{n,k}$, $j = -n/2, \ldots, n/2-1$, its closest point in the set $\Lambda_{n,k}$. Denote this subset of $\Lambda_{n,k}$ by $\tilde{\Lambda}_{n,k}$. Then we use the algorithm presented in [10] to resample a trigonometric polynomial from the set $\tilde{\Lambda}_{n,k}$ to the set $\tilde{C}_{n,k}$ with an arbitrary prescribed accuracy ε. An important property of the set $\tilde{\Lambda}_{n,k}$ is that its points are "not too far" from the points of $\tilde{C}_{n,k}$. Specifically, if $j = -\frac{n}{2}, \ldots, -|k|-1, |k|+1, \ldots, \frac{n}{2}-1$, then $p_j \in C_{n,k}$, and hence the distance between p_j and its closest point in $\tilde{\Lambda}_{n,k}$ is zero. If $p_j \notin C_{n,k}$, then we can find a point in $\Lambda_{n,k}$ whose distance to p_j is less than $-2k/n$, which goes to zero as k goes to zero (approaches the origin).

A simple induction shows that, at the beginning of the $k+n/2$ step of Algorithm 3 ($k = -n/2, \ldots, 0$), we have already recovered the values of $\hat{I}$ on $C_{n,k}$ by using the operators $H^v_{n,q}$, with $|q| > |k|$. By combining the values of $\hat{I}$ on $C_{n,k}$ with the values of $\hat{I}$ on $\Omega_{n,k}$, which are the values of the ppFT that correspond to row k, we obtain the values of $\hat{I}$ on the set $\Lambda_{n,k}$. Hence, at step $k+n/2$ we apply $H^h_{n,k}$ on the set $\Lambda_{n,k}$ and recover the values of $\hat{I}$ on the set $\tilde{C}_{n,k}$. In other words, we recover row k of $\hat{I}_D$ (see (33)). A similar argument applies to $H^v_{n,k}$. So with the operators $H^h_{n,k}$ and $H^v_{n,k}$ we may recover $\hat{I}_D$ to accuracy ε in $O(n^2 \log n + n^2 \log(1/\varepsilon))$ operations.

7.4. Recovering I from $\hat{I}_D$. Next, we present a fast algorithm that recovers the image I from the values of $\hat{I}_D(k,l)$, $k,l=-n/2,\dots,n/2$ (given by (33)). We define $\mathcal{F}_D:\mathbb{C}^n\to\mathbb{C}^{n+1}$ as

$$(41)\qquad (\mathcal{F}_D x)(k)=\sum_{u=-n/2}^{n/2-1} x(u)e^{-2\pi iu(2k)/m},\quad k=-n/2,\dots n/2,\quad m=2n+1.$$

Given a vector x of length n, the operator $\mathcal{F}_D$ is implemented by symmetrically zero padding x to length m, applying the FFT on the padded vector, and discarding every other sample. The complexity of applying $\mathcal{F}_D$ on a vector of length n is $O(n\log n)$ operations.

From (33) we see that $\hat{I}_D$ can be computed by a separable application of the 1D operator $\mathcal{F}_D$ along the rows and columns of I. Since each application of $\mathcal{F}_D$ requires $O(n\log n)$ operations, the total complexity of computing $\hat{I}_D$ is $O(n^2\log n)$ operations. To recover I from $\hat{I}_D$ we need to apply $\mathcal{F}_D^{-1}$ along the rows and columns of $\hat{I}_D$. In the remainder of the section we show that each application of $\mathcal{F}_D^{-1}$ to a row/column of $\hat{I}_D$ requires $O(n\log n)$ operations. Hence, recovering I from $\hat{I}_D$, that is, applying $\mathcal{F}_D^{-1}$ to all rows and columns of $\hat{I}_D$, requires $O(n^2\log n)$ operations.

We start with the adjoint operator $\mathcal{F}_D^*$. Let y be a vector of length $n+1$. The operator $\mathcal{F}_D^*$ is defined by

$$(42)\quad (\mathcal{F}_D^* y)(u)=\sum_{k=-n/2}^{n/2} y(k)e^{2\pi iu(2k)/m},\quad u=-n/2,\dots,n/2-1,\quad m=2n+1.$$

It is easy to verify that $\mathcal{F}_D^*$ is indeed the adjoint of $\mathcal{F}_D$.

The application of $\mathcal{F}_D^*$ to y is computed by inserting a zero between every two elements of y, resulting in a vector of length m, applying the adjoint Fourier transform, which is the inverse FFT multiplied by m, and retaining the n central elements. Clearly, the application of $\mathcal{F}_D^*$ to y requires $O(n\log n)$ operations. The operator $\mathcal{F}_D$ is not unitary, so the adjoint operator $\mathcal{F}_D^*$ is not the inverse of $\mathcal{F}_D$.

Applying the operator $\mathcal{F}_D^{-1}$ on a vector y is equivalent to solving the linear system $\mathcal{F}_D x=y$. We apply $\mathcal{F}_D^*$ on both sides and obtain the normal equations $\mathcal{F}_D^*\mathcal{F}_D x=\mathcal{F}_D^* y$ or, equivalently, $x=(\mathcal{F}_D^*\mathcal{F}_D)^{-1}\mathcal{F}_D^* y$. Solving the normal equations gives the solution to $\min_x\|\mathcal{F}_D x-y\|_2$. Hence, if y is not in the range of $\mathcal{F}_D$, the inversion algorithm finds the vector x such that $\mathcal{F}_D x$ is closest to y.

Note that $\mathcal{F}_D^*\mathcal{F}_D$ is invertible. To see this, first note that $x(u)$, $u=-n/2,\dots,n/2-1$, in (41) is uniquely determined by the samples $(\mathcal{F}_D x)(k)$, $k=-n/2,\dots n/2$. Therefore, $\operatorname{Ker}\mathcal{F}_D=\{0\}$. Next, $\mathcal{F}_D^*\mathcal{F}_D$ is positive definite. Indeed, for an arbitrary vector x

$$\langle \mathcal{F}_D^*\mathcal{F}_D x,x\rangle=\langle \mathcal{F}_D x,\mathcal{F}_D x\rangle=\|\mathcal{F}_D x\|^2\geq 0,$$

but, since $\operatorname{Ker}\mathcal{F}_D=\{0\}$, the last equation is strictly positive and $\mathcal{F}_D^*\mathcal{F}_D$ is positive definite and invertible.

The matrix $\mathcal{F}_D^*\mathcal{F}_D$ is a Toeplitz matrix, whose entries are given by

$$(\mathcal{F}_D^*\mathcal{F}_D)_{k,l}=\sum_{u=-n/2}^{n/2} e^{\frac{4\pi iu}{2n+1}(k-l)},\quad k,l=-n/2,\dots,n/2-1.$$

Moreover, since $\mathcal{F}_D^*\mathcal{F}_D$ is symmetric and positive definite, x_0 in the Gohberg–Semencul decomposition (32) is positive [27]. Therefore, as shown in section 7.1, applying

Table 3

Inverting the ppFT of a Gaussian bump with $\varepsilon = 10^{-7}$. Column 1: n, size of $n \times n$ image. Column 2: E_2, relative ℓ_2 reconstruction error. Column 3: E_∞, relative ℓ_∞ reconstruction error. Column 4: t_{Fwd}, time in seconds to compute forward ppFT. Column 5: t_{Inv}, time in seconds to compute inverse ppFT using Algorithm 3.

n	E_2	E_∞	t_{Fwd}	t_{Inv}
8	1.54826e-013	1.42780e-013	0.062	0.281
16	8.46571e-013	5.40734e-013	0.031	0.047
32	2.30805e-012	2.17171e-012	0.062	0.203
64	1.25906e-012	1.49238e-012	0.156	0.703
128	7.24066e-013	7.32485e-013	0.484	3.702
256	4.32719e-013	4.99887e-013	1.906	18.462
512	2.49692e-013	2.92489e-013	7.435	89.174

Table 4

Inverting the ppFT of a noise image with $\varepsilon = 10^{-5}$. Legend as in Table 3.

n	E_2	E_∞	t_{Fwd}	t_{Inv}
8	2.94094e-009	4.31691e-009	0.016	0.016
16	7.40180e-009	1.11551e-008	0.016	0.031
32	3.00908e-008	5.76409e-008	0.062	0.110
64	2.28288e-008	3.79261e-008	0.141	0.578
128	1.47706e-008	3.01046e-008	0.515	3.000
256	1.06168e-008	2.58128e-008	1.860	15.390
512	8.40374e-009	1.98289e-008	7.328	73.938

$(\mathcal{F}_D^*\mathcal{F}_D)^{-1}$ on an arbitrary vector requires $O(n \log n)$ operations. Since application of $\mathcal{F}_D^*$ requires also $O(n \log n)$ operations, computing $x = (\mathcal{F}_D^*\mathcal{F}_D)^{-1}\mathcal{F}_D^* y$ for an arbitrary y requires $O(n \log n)$ operations.

We recover the image I by applying $(\mathcal{F}_D^*\mathcal{F}_D)^{-1}\mathcal{F}_D^*$ on all rows and columns of $\hat{I}_D$. Since each application requires $O(n \log n)$ operations, the total complexity of recovering I from $\hat{I}_D$ is $O(n^2 \log n)$ operations.

7.5. Numerical results. Algorithm 3 was implemented in MATLAB and applied to two types of test images of various sizes. The first image is a Gaussian bump of size $n \times n$ with mean $\mu_x = \mu_y = 0$ and standard deviation $\sigma_x = \sigma_y = \frac{n}{6}$. The second is a noise image whose entries are uniformly distributed in $[0, 1]$. For each test image we compute its ppFT followed by inverse ppFT (Algorithm 3). The reconstructed image is then compared to the original image. We use the error measures given by (31).

Results are summarized in Tables 3–6. Table 3 presents the results of inverting the ppFT of a Gaussian bump. Tables 4–6 present the results of inverting the ppFT of a noise image, whose entries are uniformly distributed in $[0, 1]$, for various values of ε. All tests were implemented in MATLAB on a Pentium 2.8 GHz running Linux. As we see from Tables 3–6, the actual accuracy is higher than the prescribed one. The reason for this is that the error bounds in [10] hold for any sampling geometry. In the special sampling geometry involved in the inversion of the ppFT, these estimates are too pessimistic; note that for the noise image the actual accuracy is consistently three digits more accurate than the prescribed accuracy.

8. Conclusions.

We described the pp grid in frequency space and the associated ppFT, a fast algorithm to evaluate the Fourier transform on the pp grid in closed form. We proved correctness of the algorithm, showed that the transform is invertible, and presented two inversion algorithms.

TABLE 5
Inverting the ppFT of a noise image with $\varepsilon = 10^{-7}$. Legend as in Table 3.

n	E_2	E_∞	t_{Fwd}	t_{Inv}
8	1.52637e-011	2.30025e-011	0.015	0.016
16	5.16820e-011	6.90594e-011	0.031	0.031
32	1.85304e-010	2.67004e-010	0.047	0.125
64	1.16524e-010	1.66030e-010	0.141	0.656
128	6.71729e-011	1.25607e-010	0.500	3.297
256	5.53006e-011	1.16686e-010	1.844	16.875
512	3.94900e-011	9.00832e-011	7.313	81.468

TABLE 6
Inverting the ppFT of a noise image with $\varepsilon = 10^{-11}$. Legend as in Table 3.

n	E_2	E_∞	t_{Fwd}	t_{Inv}
8	1.30958e-015	1.34194e-015	0.000	0.016
16	1.67241e-015	2.24504e-015	0.032	0.031
32	6.50428e-015	1.10842e-014	0.047	0.125
64	1.59849e-014	2.29404e-014	0.156	0.735
128	3.70890e-014	6.79917e-014	0.500	3.875
256	7.27812e-014	1.77150e-013	1.860	19.781
512	3.41732e-013	6.84542e-013	7.250	95.671

Both the forward and the inverse transforms can be generalized to higher dimensions. In particular, the direct inversion algorithm is based only on 1D operations, and so its generalization to higher dimensions is relatively straightforward. The key difference as we move to higher dimensions is that the condition $m = 2n + 1$ must be replaced by a different condition for each dimension; for example, $m = 3n + 1$ will work in dimension 3.

The ppFT is applicable to problems that require polar Fourier representations but whose discretizations need not be uniform. Examples of such applications are image registration [24], symmetry detection [23], and spiral Fourier transform [28]. The ppFT is also closely related to the discrete Radon transform [3]. Like the continuous Radon transform, the discrete Radon transform is related to the Fourier transform of the underlying object through the projection-slice theorem. Thus, the ppFT provides an efficient algorithm and an infrastructure for the computation and inversion of the discrete Radon transform. See the companion article [3].

REFERENCES

[1] E. ARIAS-CASTRO, D. L. DONOHO, AND X. HUO, *Near-optimal detection of geometric objects by fast multiscale methods*, IEEE Trans. Inform. Theory, 51 (2005), pp. 2402–2425.

[2] A. AVERBUCH, R. COIFMAN, D.L. DONOHO, M. ELAD, AND M. ISRAELI, *Fast and accurate polar Fourier transform*, Appl. Comput. Harmon. Anal., 21 (2006), pp. 145–167.

[3] A. AVERBUCH, R. R. COIFMAN, D. L. DONOHO, M. ISRAELI, Y. SHKOLNISKY, AND I. SEDELNIKOV, *A framework for discrete integral transformations* II—*The* 2*D discrete Radon transform*, SIAM J. Sci. Comput., 30 (2008), pp. 785–803.

[4] A. AVERBUCH, R. R. COIFMAN, D. L. DONOHO, M. ISRAELI, J. WALDÉN, AND Y. SHKOLNISKY, *Fast Slant Stack: A Notion of Radon Transform for Data in a Cartesian Grid which is Rapidly Computible, Algebraically Exact, Geometrically Faithful and Invertible*, manuscript, 2001.

[5] D. H. BAILEY AND P. N. SWARZTRAUBER, *The fractional Fourier transform and applications*, SIAM Rev., 33 (1991), pp. 389–404.

[6] E. CANDÈS, L. DEMANET, D. DONOHO, AND L. YING, *Fast discrete curvelet transforms*, Multiscale Model. Simul., 5 (2006), pp. 861–899.

[7] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[8] D. L. Donoho and A. G. Flesia, *Digital ridgelet transform based on true ridge functions*, in Beyond Wavelets, Vol. 10, J. Stoecker and G. V. Welland, eds., Academic Press, New York, 2003.

[9] D. L. Donoho and O. Levi, *Fast x-ray and beamlet transforms for three-dimensional data*, in Modern Signal Processing, Math. Sci. Res. Inst. Publ. 46, D. M. Healy and D. Rockmore, eds., Cambridge University Press, Cambridge, UK, 2004, pp. 79–116.

[10] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, II, Appl. Comput. Harmon. Anal., 2 (1995), pp. 85–100.

[11] P. Edholm and G. T. Herman, *Linograms in image reconstruction from projections*, IEEE Trans. Med. Imaging, 6 (1987), pp. 301–307.

[12] P. Edholm, G. T. Herman, and D. A. Roberts, *Image reconstruction from linograms: Implementation and evaluation*, IEEE Trans. Med. Imaging, 7 (1988), pp. 239–246.

[13] H. G. Feichtinger, K. Gröchenig, and T. Strohmer, *Efficient numerical methods in non-uniform sampling theory*, Numer. Math., 69 (1995), pp. 423–440.

[14] K. Fourmont, *Non-equispaced fast Fourier transforms with applications to tomography*, J. Fourier Anal. Appl., 9 (2003), pp. 431–450.

[15] I. Gohberg and V. Olshevsky, *Fast algorithms with preprocessing for matrix-vector multiplication problems*, J. Complexity, 10 (1994), pp. 411–427.

[16] I. Gohberg and A. Semencul, *The inversion of finite Toeplitz matrices and their continual analogues*, Mat. Issled., 7 (1972), pp. 201–223 (in Russian).

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1984.

[18] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[19] L. Greengard and J.-Y. Lee, *Accelerating the nonuniform fast Fourier transform*, SIAM Rev., 46 (2004), pp. 443–454.

[20] http://curvelet.org/.

[21] http://www-stat.stanford.edu/∼beamlab/.

[22] T. Kailath and A. H. Sayed, eds., *Fast Reliable Algorithms for Matrices with Structure*, SIAM, Philadelphia, 1999.

[23] Y. Keller and Y. Shkolnisky, *A signal processing approach to symmetry detection*, IEEE Trans. Image Process., 15 (2006), pp. 2198–2207.

[24] Y. Keller, Y. Shkolnisky, and A. Averbuch, *The angular difference function and its application to image registration*, IEEE Trans. Pattern Anal. Mach. Intell., 27 (2005), pp. 969–976.

[25] W. Lawton, *A new polar Fourier transform for computer-aided tomography and spotlight synthetic aperture radar*, IEEE Trans. Acoust. Speech Signal Process., 36 (1988), pp. 931–933.

[26] O. Levi, *Multiscale Geometric Analysis of Three-Dimensional Data*, Ph.D. thesis, Stanford University, Stanford, CA, 2005.

[27] E. Linzer and M. Vetterli, *Iterative Toeplitz solvers with local quadratic convergence*, Computing, 49 (1993), pp. 339–347.

[28] M. Lustig, J. Tsaig, J. H. Lee, and D. Donoho, *Fast spiral Fourier transform for iterative MR image reconstruction*, in Proceedings of the IEEE International Symposium on Biomedical Imaging: Nano to Macro, Arlington, VA, 2004, pp. 784–787.

[29] R. M. Mersereau and A. V. Oppenheim, *Digital reconstruction of multidimensional signals from their projections*, Proc. IEEE, 62 (1974), pp. 1319–1338.

[30] F. Natterer, *The Mathematics of Computerized Tomography*, Classics Appl. Math. 32, SIAM, Philadelphia, 2001.

[31] J. E. Pasciak, *A Note on the Fourier Algorithm for Image Reconstruction*, preprint, Applied Mathematics Department, Brookhaven National Laboratory, Upton, NY, 1973.

[32] D. Potts and G. Steidl, *A new linogram algorithm for computerized tomography*, IMA J. Numer. Anal., 21 (2001), pp. 769–782.

[33] L. R. Rabiner, R. W. Schafer, and C. M. Rader, *The chirp Z-transform algorithm*, IEEE Trans. Audio Electroacoust., 17 (1969), pp. 86–92.

[34] J. L. Starck, E. J. Candes, and D. L. Donoho, *The curvelet transform for image denoising*, IEEE Trans. Image Process., 11 (2002), pp. 670–684.

[35] A. F. Ware, *Fast approximate Fourier transforms for irregularly spaced data*, SIAM Rev., 40 (1998), pp. 838–856.